In [ ]:
```
# ### uncommnet these if this is the first time you use these packages ####
# !pip install pandas
# !pip install numpy
# !pip install scikit-learn
# !pip install seaborn
# !pip install pyod
# !pip install PiML
```

In [ ]:
```
# mute warnings
import warnings
warnings.filterwarnings('ignore')

import sklearn as sk
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats
import statsmodels.api as sm
```

Data set

We're going to use the Motor Trends Cars ("mtcars") data set that is built into the R programming language.

```
mpg - Miles per Gallon
cyl - # of cylinders
disp - displacement, in cubic inches
hp - horsepower
drat - driveshaft ratio
wt - weight
qsec - 1/4 mile time; a measure of acceleration
vs - 'V' or straight - engine shape
am - transmission; auto or manual
gear - # of gears
carb - # of carburetors.
```

In [ ]:
```
# Load the dataset
df = pd.read_csv("mtcars.csv")

# Here we are going to use the "model" of the car as a the index to our dataframe
df.set_index('model', inplace=True)
df.head()
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [ ]:
```
# Descriptive statistics
df.describe()
```

Out[ ]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.0000
mean	20.090625	6.187500	230.721875	146.687500	3.596563	3.217250	17.848750	0.437500	0.406250	3.687500	2.8125
std	6.026948	1.785922	123.938694	68.562868	0.534679	0.978457	1.786943	0.504016	0.498991	0.737804	1.6152
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.0000
25%	15.425000	4.000000	120.825000	96.500000	3.080000	2.581250	16.892500	0.000000	0.000000	3.000000	2.0000
50%	19.200000	6.000000	196.300000	123.000000	3.695000	3.325000	17.710000	0.000000	0.000000	4.000000	2.0000
75%	22.800000	8.000000	326.000000	180.000000	3.920000	3.610000	18.900000	1.000000	1.000000	4.000000	4.0000
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.000000	1.000000	5.000000	8.0000

In [ ]:
```
# Check whether there are any missing values
df.isnull().sum()
```

Out[ ]:

```
mpg     0
cyl     0
disp    0
hp      0
drat    0
wt      0
qsec    0
vs      0
am      0
gear    0
carb    0
dtype: int64
```

In [ ]:

```python
# Heatmap based on standardized values
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df)
scaled = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled, columns=df.columns, index=df.index)
# scaled_df.abs().sum(axis=1)
df.style.background_gradient(cmap ='coolwarm', gmap=scaled_df, axis=None, vmin=-3, vmax=3)
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.000000	6	160.000000	110	3.900000	2.620000	16.460000	0	1	4	4
Mazda RX4 Wag	21.000000	6	160.000000	110	3.900000	2.875000	17.020000	0	1	4	4
Datsun 710	22.800000	4	108.000000	93	3.850000	2.320000	18.610000	1	1	4	1
Hornet 4 Drive	21.400000	6	258.000000	110	3.080000	3.215000	19.440000	1	0	3	1
Hornet Sportabout	18.700000	8	360.000000	175	3.150000	3.440000	17.020000	0	0	3	2
Valiant	18.100000	6	225.000000	105	2.760000	3.460000	20.220000	1	0	3	1
Duster 360	14.300000	8	360.000000	245	3.210000	3.570000	15.840000	0	0	3	4
Merc 240D	24.400000	4	146.700000	62	3.690000	3.190000	20.000000	1	0	4	2
Merc 230	22.800000	4	140.800000	95	3.920000	3.150000	22.900000	1	0	4	2
Merc 280	19.200000	6	167.600000	123	3.920000	3.440000	18.300000	1	0	4	4
Merc 280C	17.800000	6	167.600000	123	3.920000	3.440000	18.900000	1	0	4	4
Merc 450SE	16.400000	8	275.800000	180	3.070000	4.070000	17.400000	0	0	3	3
Merc 450SL	17.300000	8	275.800000	180	3.070000	3.730000	17.600000	0	0	3	3
Merc 450SLC	15.200000	8	275.800000	180	3.070000	3.780000	18.000000	0	0	3	3
Cadillac Fleetwood	10.400000	8	472.000000	205	2.930000	5.250000	17.980000	0	0	3	4
Lincoln Continental	10.400000	8	460.000000	215	3.000000	5.424000	17.820000	0	0	3	4
Chrysler Imperial	14.700000	8	440.000000	230	3.230000	5.345000	17.420000	0	0	3	4
Fiat 128	32.400000	4	78.700000	66	4.080000	2.200000	19.470000	1	1	4	1
Honda Civic	30.400000	4	75.700000	52	4.930000	1.615000	18.520000	1	1	4	2
Toyota Corolla	33.900000	4	71.100000	65	4.220000	1.835000	19.900000	1	1	4	1
Toyota Corona	21.500000	4	120.100000	97	3.700000	2.465000	20.010000	1	0	3	1
Dodge Challenger	15.500000	8	318.000000	150	2.760000	3.520000	16.870000	0	0	3	2
AMC Javelin	15.200000	8	304.000000	150	3.150000	3.435000	17.300000	0	0	3	2
Camaro Z28	13.300000	8	350.000000	245	3.730000	3.840000	15.410000	0	0	3	4
Pontiac Firebird	19.200000	8	400.000000	175	3.080000	3.845000	17.050000	0	0	3	2
Fiat X1-9	27.300000	4	79.000000	66	4.080000	1.935000	18.900000	1	1	4	1
Porsche 914-2	26.000000	4	120.300000	91	4.430000	2.140000	16.700000	0	1	5	2
Lotus Europa	30.400000	4	95.100000	113	3.770000	1.513000	16.900000	1	1	5	2
Ford Pantera L	15.800000	8	351.000000	264	4.220000	3.170000	14.500000	0	1	5	4
Ferrari Dino	19.700000	6	145.000000	175	3.620000	2.770000	15.500000	0	1	5	6
Maserati Bora	15.000000	8	301.000000	335	3.540000	3.570000	14.600000	0	1	5	8
Volvo 142E	21.400000	4	121.000000	109	4.110000	2.780000	18.600000	1	1	4	2

Use regression models to predict fuel comsumption(mpg)

In [ ]:
```
from sklearn import  linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

split the dataset into training and test sets

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    df.iloc[:,1:], df.iloc[:,0], test_size=0.1, random_state=42)
```

Linear Regression

In [ ]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)
print(X_train.columns)
print(regr.coef_)
#Predict using the test set
pred = regr.predict(X_test)

#Calculate the metrics for regression
reg_r2 = r2_score(y_test, pred)
reg_mse = mean_squared_error(y_test, pred)
print('------------')
print(reg_r2, reg_mse)
```

```
Index(['cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am', 'gear', 'carb'], dtype='object')
[-0.40110818  0.01316612 -0.02167558  0.59836675 -3.83000087  0.70898298
  0.07163707  1.57941842  0.659154    0.0778369 ]
------------
0.8383955580471785 9.935744099588128
```

SVM for regression

In [ ]:
```
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
# Create svm regression object
regr_svm = make_pipeline(StandardScaler(), SVR(kernel='linear',C=1.0, epsilon=0.2))

# Train the model using the training sets
regr_svm.fit(X_train, y_train)
```

Out[ ]:
```
▸ Pipeline
▸ StandardScaler
▸ SVR
```

In [ ]:
```
#Predict using the test set
pred = regr_svm.predict(X_test)

#Calculate the metrics for regression
svm_r2 = r2_score(y_test, pred)
svm_mse = mean_squared_error(y_test, pred)
```

In [ ]:
```
print(svm_r2, svm_mse)
```

```
0.784034144131135 13.27798575479757
```

Gradient Boosting for regression

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [ ]:
```
# Create gradient boosting regression object
reg_gb = GradientBoostingRegressor(random_state=0)

# Train the model using the training sets
reg_gb.fit(X_train, y_train)
```

Out[ ]:
```
▾ GradientBoostingRegressor
GradientBoostingRegressor(random_state=0)
```

In [ ]:
```python
#Predict using the test set
pred = reg_gb.predict(X_test)

#Calculate the metrics for regression
gb_r2 = r2_score(y_test, pred)
gb_mse = mean_squared_error(y_test, pred)
```

In [ ]:
```python
#Make a table to compare the performance of different models
pd.DataFrame({'LinearRegression':[reg_r2,reg_mse], 'SVM':[svm_r2, svm_mse],  'GradientBoosting':[gb_r2, gb_mse]
```

Out[ ]:

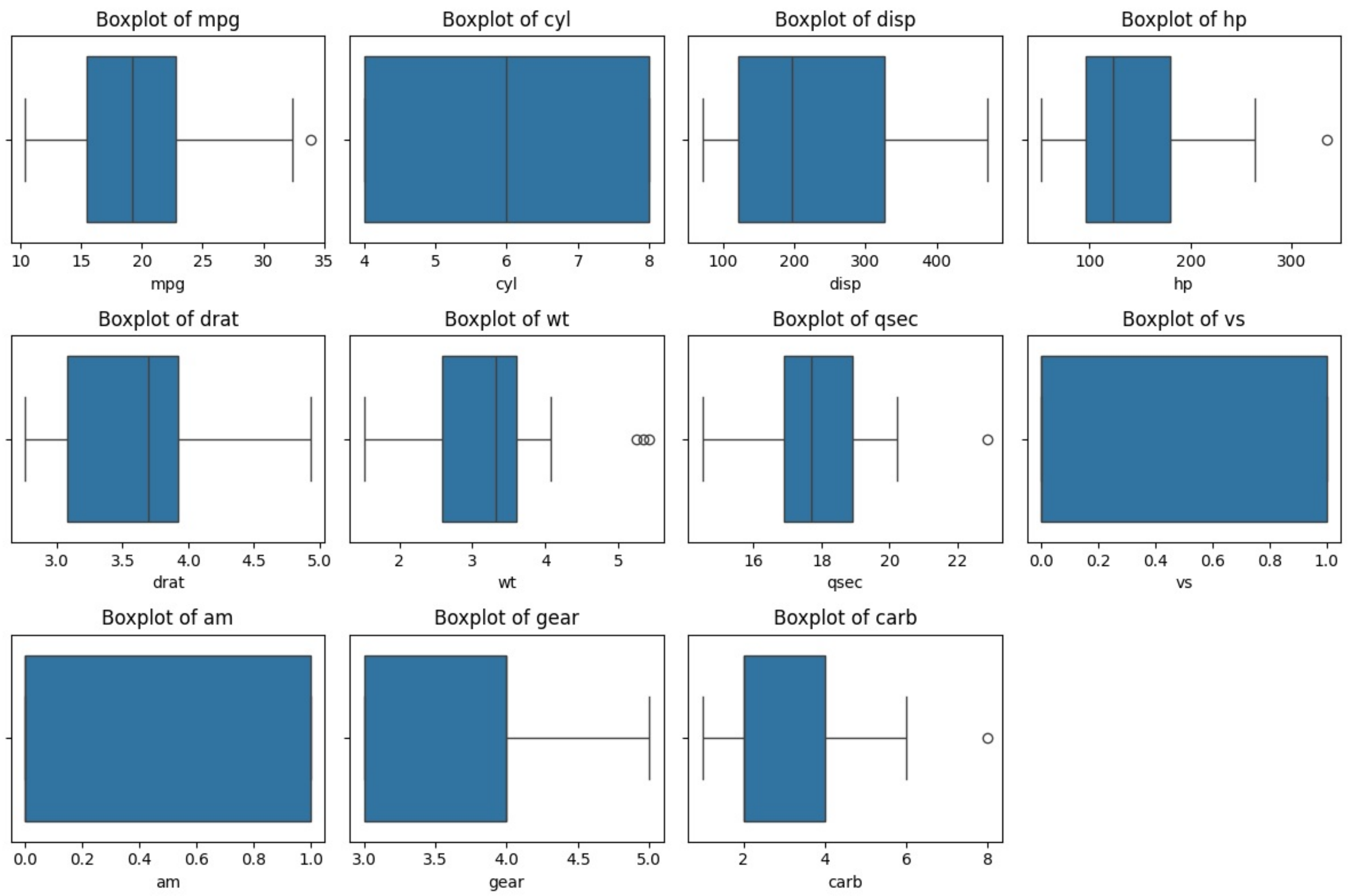
	LinearRegression	SVM	GradientBoosting
R2	0.838396	0.784034	0.876541
MSE	9.935744	13.277986	7.590488

Outlier Detection: univariate case

In [ ]:
```python
plt.figure(figsize=(12, 8))


for i, col in enumerate(df.columns):
    plt.subplot(3, 4, i+1)  # Create a 3x4 grid of subplots
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')

plt.tight_layout()
plt.show()
```



In [ ]:
```python
# Find the outliers for "wt"
wtQ1 = df['wt'].quantile(0.25)
wtQ3 = df['wt'].quantile(0.75)
wtIQR = wtQ3 - wtQ1    #IQR is interquartile range.
print(wtQ1, wtQ3, wtIQR)

wt_upper_limit = (wtQ3 + 1.5 * wtIQR)
wt_lower_limit = (wtQ1 - 1.5 * wtIQR)
# Show the boxplot outliers
df.loc[(df['wt'] < wt_lower_limit) | (df['wt'] > wt_upper_limit)]
```

```
2.58125 3.61 1.02875
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

In [ ]:

```
# Find the outliers for "hp" and "qsec"
# hp
hpQ1 = df['hp'].quantile(0.25)
hpQ3 = df['hp'].quantile(0.75)
hpIQR = hpQ3 - hpQ1    #IQR is interquartile range.
print(hpQ1, hpQ3, hpIQR)

hp_upper_limit = (hpQ3 + 1.5 * hpIQR)
hp_lower_limit = (hpQ1 - 1.5 * hpIQR)
# Show the boxplot outliers
df.loc[(df['hp'] < hp_lower_limit) | (df['hp'] > hp_upper_limit)]
```

```
96.5 180.0 83.5
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Maserati Bora	15.0	8	301.0	335	3.54	3.57	14.6	0	1	5	8

In [ ]:

```
#qsec
qsecQ1 = df['qsec'].quantile(0.25)
qsecQ3 = df['qsec'].quantile(0.75)
qsecIQR = qsecQ3 - qsecQ1    #IQR is interquartile range.
print(qsecQ1, qsecQ3, qsecIQR)

qsec_upper_limit = (qsecQ3 + 1.5 * qsecIQR)
qsec_lower_limit = (qsecQ1 - 1.5 * qsecIQR)
# Show the boxplot outliers
df.loc[(df['qsec'] < qsec_lower_limit) | (df['qsec'] > qsec_upper_limit)]
```

```
16.8925 18.9 2.0075000000000003
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2

Outlier Treatments

Capping Outliers (using IQR)

In [ ]:

```
df2 = X_train.copy()
```

In [ ]:

```
# This code will "cap" (or floor) the outliers to our limit for the wt predictor
df2['wt'] = np.where(df2['wt'] > wt_upper_limit,
  wt_upper_limit,
  np.where(
    df2['wt'] < wt_lower_limit,
    wt_lower_limit,
    df2['wt']
   )
)
```

In [ ]:

```
# Write code to cap/floor the hp and qsec predictors
# hp
df2['hp'] = np.where(df2['hp'] > hp_upper_limit,
  hp_upper_limit,
  np.where(
    df2['hp'] < hp_lower_limit,
    hp_lower_limit,
    df2['hp']
   )
)
```

In [ ]:

```
#qsec
df2['qsec'] = np.where(df2['qsec'] > qsec_upper_limit,
  qsec_upper_limit,
  np.where(
    df2['qsec'] < qsec_lower_limit,
```

```
    qsec_lower_limit,
    df2['qsec']
   )
)
```

In [ ]:
```
# Use describe to ensure our min/max looks right
df2.describe()
```

Out[ ]:

	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000	28.000000
mean	6.142857	224.978571	144.044643	3.618214	3.158009	17.868973	0.464286	0.392857	3.678571	2.750000
std	1.799471	116.042580	67.338698	0.544066	0.900980	1.733991	0.507875	0.497347	0.722832	1.554563
min	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.000000
25%	4.000000	120.825000	96.500000	3.132500	2.581250	16.892500	0.000000	0.000000	3.000000	2.000000
50%	6.000000	196.300000	118.000000	3.715000	3.325000	17.790000	0.000000	0.000000	4.000000	2.000000
75%	8.000000	307.500000	180.000000	3.920000	3.570000	18.900000	1.000000	1.000000	4.000000	4.000000
max	8.000000	472.000000	305.250000	4.930000	5.153125	21.911250	1.000000	1.000000	5.000000	8.000000

In [ ]:
```
########### Linear Regression ############
regr = linear_model.LinearRegression()
regr.fit(df2, y_train)
pred = regr.predict(X_test)
reg_r2_capped = r2_score(y_test, pred)
reg_mse_capped = mean_squared_error(y_test, pred)

#################### SVM ####################
regr_svm = make_pipeline(StandardScaler(), SVR(kernel='linear',C=1.0, epsilon=0.2))
regr_svm.fit(df2, y_train)
pred = regr_svm.predict(X_test)
svm_r2_capped = r2_score(y_test, pred)
svm_mse_capped = mean_squared_error(y_test, pred)

#################### Gradient Boosting ####################
reg_gb = GradientBoostingRegressor(random_state=0)
reg_gb.fit(df2, y_train)
pred = reg_gb.predict(X_test)
gb_r2_capped = r2_score(y_test, pred)
gb_mse_capped = mean_squared_error(y_test, pred)
```

In [ ]:
```
# compare metrics: no treatment VS "capped"
pd.DataFrame(
    {'LinearRegression':[reg_r2,reg_r2_capped],
        'SVM':[svm_r2, svm_r2_capped],
        'GradientBoosting':[gb_r2, gb_r2_capped]},
    index=['R2', 'R2_capped'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
R2	0.838396	0.784034	0.876541
R2_capped	0.840594	0.780404	0.880187

In [ ]:
```
pd.DataFrame(
    {'LinearRegression':[reg_mse,reg_mse_capped],
     'SVM':[svm_mse, svm_mse_capped],
     'GradientBoosting':[gb_mse, gb_mse_capped]},
    index=['MSE', 'MSE_capped'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
MSE	9.935744	13.277986	7.590488
MSE_capped	9.800564	13.501195	7.366320

Removing rows with outliers

In [ ]:
```
df3 = X_train.copy()
```

In [ ]:
```
# This code will remove outliers beyond our limit for the wt predictor
y_train.drop(y_train[(df3.wt < wt_lower_limit) | (df3.wt > wt_upper_limit)].index, inplace=True)
df3.drop(df3[df3.wt < wt_lower_limit].index, inplace=True)
df3.drop(df3[df3.wt > wt_upper_limit].index, inplace=True)
```

In [ ]:
```
# Write code to remove outliers beyond our limit for the hp and qsec predictors
```

```python
#hp
y_train.drop(y_train[(df3.hp < hp_lower_limit) | (df3.hp > hp_upper_limit)].index, inplace=True)
df3.drop(df3[df3.hp < hp_lower_limit].index, inplace=True)
df3.drop(df3[df3.hp > hp_upper_limit].index, inplace=True)
```

In [ ]:
```python
#qsec
y_train.drop(y_train[(df3.qsec < qsec_lower_limit) | (df3.qsec > qsec_upper_limit)].index, inplace=True)
df3.drop(df3[df3.qsec < qsec_lower_limit].index, inplace=True)
df3.drop(df3[df3.qsec > qsec_upper_limit].index, inplace=True)
```

In [ ]:
```python
# Use describe to ensure our min/max looks right
df3.describe()
```

Out[ ]:

	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
count	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000	24.000000
mean	6.000000	206.066667	133.250000	3.653750	2.974917	17.850833	0.500000	0.416667	3.666667	2.458333
std	1.769303	101.054405	59.027812	0.561069	0.751299	1.531319	0.510754	0.503610	0.701964	1.178767
min	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.000000	0.000000	3.000000	1.000000
25%	4.000000	120.250000	96.000000	3.132500	2.428750	16.892500	0.000000	0.000000	3.000000	1.750000
50%	6.000000	167.600000	111.500000	3.750000	3.202500	17.800000	0.500000	0.000000	4.000000	2.000000
75%	8.000000	282.850000	176.250000	3.960000	3.475000	18.900000	1.000000	1.000000	4.000000	4.000000
max	8.000000	360.000000	264.000000	4.930000	4.070000	20.220000	1.000000	1.000000	5.000000	4.000000

In [ ]:
```python
########### Linear Regression ############
regr = linear_model.LinearRegression()
regr.fit(df3, y_train)
pred = regr.predict(X_test)
reg_r2_removed = r2_score(y_test, pred)
reg_mse_removed = mean_squared_error(y_test, pred)

#################### SVM #####################
regr_svm = make_pipeline(StandardScaler(), SVR(kernel='linear',C=1.0, epsilon=0.2))
regr_svm.fit(df3, y_train)
pred = regr_svm.predict(X_test)
svm_r2_removed = r2_score(y_test, pred)
svm_mse_removed = mean_squared_error(y_test, pred)

#################### Gradient Boosting ####################
reg_gb = GradientBoostingRegressor(random_state=0)
reg_gb.fit(df3, y_train)
pred = reg_gb.predict(X_test)
gb_r2_removed = r2_score(y_test, pred)
gb_mse_removed = mean_squared_error(y_test, pred)
```

In [ ]:
```python
# compare metrics: no treatment VS "capped" VS "removed"
pd.DataFrame(
    {'LinearRegression':[reg_r2,reg_r2_capped,reg_r2_removed],
        'SVM':[svm_r2, svm_r2_capped, svm_r2_removed],
        'GradientBoosting':[gb_r2, gb_r2_capped, gb_r2_removed]},
    index=['R2', 'R2_capped', 'R2_removed'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
R2	0.838396	0.784034	0.876541
R2_capped	0.840594	0.780404	0.880187
R2_removed	0.714471	0.782641	0.722095

In [ ]:
```python
pd.DataFrame(
    {'LinearRegression':[reg_mse,reg_mse_capped,reg_mse_removed],
     'SVM':[svm_mse, svm_mse_capped, svm_mse_removed],
     'GradientBoosting':[gb_mse, gb_mse_capped, gb_mse_removed]},
    index=['MSE', 'MSE_capped', 'MSE_removed'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
MSE	9.935744	13.277986	7.590488
MSE_capped	9.800564	13.501195	7.366320
MSE_removed	17.554841	13.363656	17.086114

Outlier Detection: multivariate case

In [ ]:
```
# If not already installed, install pyod
!pip install pyod
```

```
Requirement already satisfied: pyod in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-pa
ckages (2.0.1)
Requirement already satisfied: joblib in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/site-
packages (from pyod) (1.4.2)
Requirement already satisfied: matplotlib in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/s
ite-packages (from pyod) (3.7.5)
Requirement already satisfied: numpy>=1.19 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/
site-packages (from pyod) (1.23.5)
Requirement already satisfied: numba>=0.51 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/
site-packages (from pyod) (0.56.4)
Requirement already satisfied: scipy>=1.5.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10
/site-packages (from pyod) (1.10.1)
Requirement already satisfied: scikit-learn>=0.22.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/py
thon3.10/site-packages (from pyod) (1.3.2)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/
lib/python3.10/site-packages (from numba>=0.51->pyod) (0.39.1)
Requirement already satisfied: setuptools in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/s
ite-packages (from numba>=0.51->pyod) (65.5.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/py
thon3.10/site-packages (from scikit-learn>=0.22.0->pyod) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python
3.10/site-packages (from matplotlib->pyod) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10
/site-packages (from matplotlib->pyod) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/pytho
n3.10/site-packages (from matplotlib->pyod) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/pytho
n3.10/site-packages (from matplotlib->pyod) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3
.10/site-packages (from matplotlib->pyod) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.1
0/site-packages (from matplotlib->pyod) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python
3.10/site-packages (from matplotlib->pyod) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/py
thon3.10/site-packages (from matplotlib->pyod) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /Users/jakebrulato/Documents/GitHub/DSBA6010/.venv/lib/python3.10/sit
e-packages (from python-dateutil>=2.7->matplotlib->pyod) (1.16.0)
```

In [ ]:
```
# use LOF (with 5 nearest neighbors) to detection multivariate outliers
# and eliminate rows with and lof score > 1.3
from pyod.models.lof import LOF

df4 = df.copy()

# Prepare the LOF model with 5 nearest neighbors
lof = LOF(n_neighbors=5)

# Fit the LOF model and predict the outliers
lof.fit(df4)
lof_scores = lof.decision_scores_

# Add the LOF scores to the dataframe
df4['LOF_Value'] = lof_scores
df4
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	LOF_Value
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1.041947
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	1.041947
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	1.024865
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	1.026977
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0.923277
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	1.432937
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	0.964717
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	1.229866
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	0.942760
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	1.127189
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	1.127728
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	0.974647
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	0.974878
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	0.974647
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	1.159139
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	1.128107
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	1.031167
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	1.173683
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	1.237937
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	1.195461
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	0.984568
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	1.030133
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	1.014205
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	0.952798
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	1.041303
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	1.167617
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	0.967056
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	0.959718
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	1.142798
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	1.587094
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	1.570968
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	1.008551

In [ ]:

```
# Sort the dataframe by the LOF_Value column in descending order
df4_sorted = df4.sort_values(by='LOF_Value', ascending=False)
df4_sorted
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	LOF_Value
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	1.587094
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	1.570968
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	1.432937
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	1.237937
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	1.229866
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	1.195461
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	1.173683
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	1.167617
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	1.159139
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	1.142798
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	1.128107
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	1.127728
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	1.127189
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1.041947
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	1.041947
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	1.041303
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	1.031167
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	1.030133
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	1.026977
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	1.024865
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	1.014205
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	1.008551
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	0.984568
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	0.974878
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	0.974647
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	0.974647
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	0.967056
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	0.964717
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	0.959718
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	0.952798
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	0.942760
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0.923277

In [ ]:

```
# # Eliminate rows with LOF score > 1.3
df4 = df4[df4['LOF_Value'] <= 1.3]
df4 = df4.sort_values(by='LOF_Value', ascending=False)
df4
```

Out[ ]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	LOF_Value
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	1.237937
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	1.229866
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	1.195461
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	1.173683
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	1.167617
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	1.159139
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	1.142798
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	1.128107
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	1.127728
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	1.127189
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	1.041947
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1.041947
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	1.041303
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	1.031167
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	1.030133
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	1.026977
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	1.024865
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	1.014205
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	1.008551
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	0.984568
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	0.974878
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	0.974647
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	0.974647
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	0.967056
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	0.964717
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	0.959718
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	0.952798
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	0.942760
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	0.923277

In [ ]:

```
df4.drop(columns=['LOF_Value'], inplace=True)
```

In [ ]:

```
# Rebuild the models
X_train_lof, X_tes_lof, y_train_lof, y_test_lof = train_test_split(
    df4.iloc[:,1:], df4.iloc[:,0], test_size=0.1, random_state=42)

########### Linear Regression ############
regr = linear_model.LinearRegression()
regr.fit(X_train_lof, y_train_lof)
print('linear')
pred = regr.predict(X_test)
print(X_train.columns)
print(regr.coef_)
reg_r2_removed_lof = r2_score(y_test, pred)
reg_mse_removed_lof = mean_squared_error(y_test, pred)
print('**********')

#################### SVM ####################
regr_svm = make_pipeline(StandardScaler(), SVR(kernel='linear',C=1.0, epsilon=0.2))
regr_svm.fit(X_train_lof, y_train_lof)
pred = regr_svm.predict(X_test)
print('SVM')
print(X_train_lof.columns)
print(regr_svm.named_steps['svr'].coef_)
svm_r2_removed_lof = r2_score(y_test, pred)
svm_mse_removed_lof = mean_squared_error(y_test, pred)
print('**********')

#################### Gradient Boosting ####################
reg_gb = GradientBoostingRegressor(random_state=0)
```

```
reg_gb.fit(X_train_lof, y_train_lof)
pred = reg_gb.predict(X_test)
print('Gradient Boosting')
print(X_train_lof.columns)
print(reg_gb.feature_importances_)
gb_r2_removed_lof = r2_score(y_test, pred)
gb_mse_removed_lof = mean_squared_error(y_test, pred)
print('**********')
```

```
linear
Index(['cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am', 'gear', 'carb'], dtype='object')
[-0.05797362  0.00461904 -0.00460456  0.34180882 -3.1441907   1.47949302
 -0.14454454  3.03273688  1.17131605 -0.65849421]
**********
SVM
Index(['cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am', 'gear', 'carb'], dtype='object')
[[-1.25811606 -0.84417163 -0.46977451  0.80524643 -1.4954124   0.50682326
   0.42413907  0.98709411 -0.21579083 -1.12926536]]
**********
Gradient Boosting
Index(['cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am', 'gear', 'carb'], dtype='object')
[1.27243682e-01 2.01626834e-01 1.44207240e-01 2.17378982e-03
 4.92413422e-01 2.55048477e-02 2.01736142e-04 2.53005140e-05
 1.68452409e-04 6.43469620e-03]
**********
```

In [ ]:

```
# compare metrics: no treatment VS "capped" VS "removed"
pd.DataFrame(
    {'LinearRegression':[reg_r2,reg_r2_capped,reg_r2_removed,reg_r2_removed_lof],
        'SVM':[svm_r2, svm_r2_capped, svm_r2_removed, svm_r2_removed_lof],
        'GradientBoosting':[gb_r2, gb_r2_capped, gb_r2_removed, gb_r2_removed_lof]},
    index=['R2', 'R2_capped', 'R2_removed', 'R2_removed_lof'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
R2	0.838396	0.784034	0.876541
R2_capped	0.840594	0.780404	0.880187
R2_removed	0.714471	0.782641	0.722095
R2_removed_lof	0.916821	0.873599	0.976491

In [ ]:

```
pd.DataFrame(
    {'LinearRegression':[reg_mse,reg_mse_capped,reg_mse_removed,reg_mse_removed_lof],
     'SVM':[svm_mse, svm_mse_capped, svm_mse_removed, svm_mse_removed_lof],
     'GradientBoosting':[gb_mse, gb_mse_capped, gb_mse_removed, gb_mse_removed_lof]},
    index=['MSE', 'MSE_capped', 'MSE_removed', 'MSE_removed_lof'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
MSE	9.935744	13.277986	7.590488
MSE_capped	9.800564	13.501195	7.366320
MSE_removed	17.554841	13.363656	17.086114
MSE_removed_lof	5.113976	7.771392	1.445399

Q1. What were the top 4 most influential features in the `regr` model above?

Depending on the answer, the most influential including negative and postive would be:

Latest All Postive/Negative LOF Regr with Outliers Removed

- wt: -3.1441907
- am: 3.03273688
- qsec: 1.47949302
- gear: 1.17131605

Latest All Postive LOF Regr with Outliers Removed:

- am: 3.03273688
- qsec: 1.47949302
- gear: 1.17131605
- drat: 0.34180882

Q2. What was the most influential features in the `regr` model above? What was the most influential feature in the `reg_gb` model above?

regr:

- wt: -3.1441907

reg_gb:

- wt: 4.92413422e-01

Q3. In the original data set (`df`) which car looks more like a *bivariate outlier* with respect to `disp` and `drat`? *Masarati Bora*, *Ford Pantera L*, or the *Toyota Corolla*?

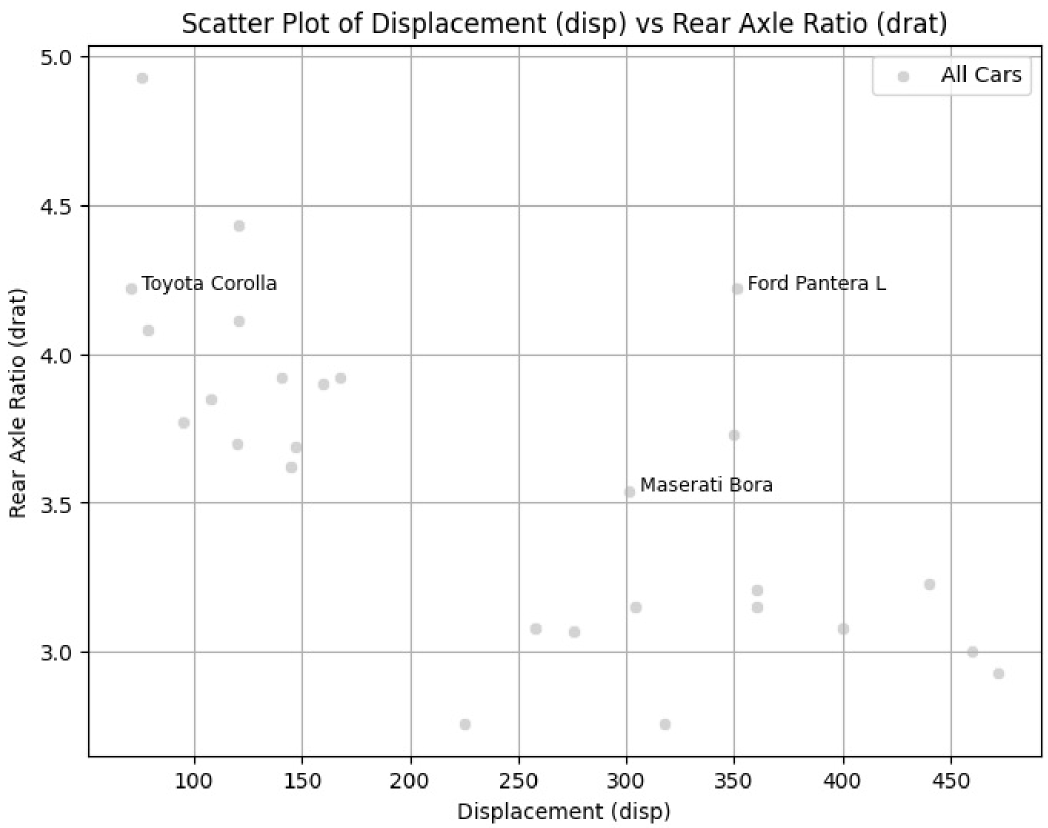
- Ford Pantera semms to be the most likely as a bivariate outlier to disp and drat. The Pantera stands out because of how far it is from the other points, Maserati is closer to most other points in the bottom right and the Corolla could be arged from but ultimately still is closer to most other points than the Pantera.

In [ ]:
```
cars_of_interest = ['Maserati Bora', 'Ford Pantera L', 'Toyota Corolla']
df_subset = df.loc[cars_of_interest, ['disp', 'drat']]

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['disp'], y=df['drat'], label='All Cars', color='lightgray')

for car, row in df_subset.iterrows():
    plt.text(row['disp'] + 5, row['drat'], car, fontsize=9)

plt.title('Scatter Plot of Displacement (disp) vs Rear Axle Ratio (drat)')
plt.xlabel('Displacement (disp)')
plt.ylabel('Rear Axle Ratio (drat)')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



Q4. Could the `scipy.stats.mstats.winsorize` function in

Python be used to easily treat *only* the outliers we found with `sns.boxplot`? Explain.

- Winsorize works by capping the certain percentile threshold across the entire distribution, it does not selectively treat specific points like we find from the IQR method. It more ideal for a uniform application across all points, not those identified as an outlier from the IQR, so I wouldn't recommend it.

Q5. How does the "capping" the outliers affect model performance? What happens if you change the `random_state` to 43 in the train/test split (and then re-build the models)? What could be done to provide more robust error metrics?

- Capping limits the values of oultiers to a certain threshold, it can create more stability and reduce the variance in the model but has its drawbacks because it can create bias and some information loss if its capped to aggresively.
- Compared to a normal state where every time the model is ran, different numbers and model performance will occur. By setting the random state to 43, you create reproducable scenario, different samples of both training and test data, and performance variability because of the new splits. When we set the split for capping to 43, the results are horrid.
- Creating a ensemble methods (Bagging and Boosting), using a bootstrap sampling technique where you sample the whole different with replacement, or cross validation which divided the data a certain amount of times (k) and trains the model 'k' times.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    df.iloc[:,1:], df.iloc[:,0], test_size=0.1, random_state=43)
```

In [ ]:
```
########### Linear Regression ############
regr = linear_model.LinearRegression()
regr.fit(df2, y_train)
pred = regr.predict(X_test)
reg_r2_capped = r2_score(y_test, pred)
reg_mse_capped = mean_squared_error(y_test, pred)

#################### SVM ####################
regr_svm = make_pipeline(StandardScaler(), SVR(kernel='linear',C=1.0, epsilon=0.2))
regr_svm.fit(df2, y_train)
pred = regr_svm.predict(X_test)
svm_r2_capped = r2_score(y_test, pred)
svm_mse_capped = mean_squared_error(y_test, pred)

#################### Gradient Boosting ####################
reg_gb = GradientBoostingRegressor(random_state=0)
reg_gb.fit(df2, y_train)
pred = reg_gb.predict(X_test)
gb_r2_capped = r2_score(y_test, pred)
gb_mse_capped = mean_squared_error(y_test, pred)
```

In [ ]:
```
# compare metrics: no treatment VS "capped"
pd.DataFrame(
    {'LinearRegression':[reg_r2,reg_r2_capped],
        'SVM':[svm_r2, svm_r2_capped],
        'GradientBoosting':[gb_r2, gb_r2_capped]},
    index=['R2', 'R2_capped'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
R2	0.838396	0.784034	0.876541
R2_capped	-1.932476	-0.300214	-5.745071

In [ ]:
```
pd.DataFrame(
    {'LinearRegression':[reg_mse,reg_mse_capped],
     'SVM':[svm_mse, svm_mse_capped],
     'GradientBoosting':[gb_mse, gb_mse_capped]},
    index=['MSE', 'MSE_capped'])
```

Out[ ]:

	LinearRegression	SVM	GradientBoosting
MSE	9.935744	13.277986	7.590488
MSE_capped	34.110190	15.123926	78.457817

Q6. *On average (given many different train/test splits)*, what modeling method is most affected by the removal of outliers in this data set? Why?

- I believe it would be anything related to a linear or logistic regression output as a removal of an outlier can affect the slope greatly. Lets say you have one extreme outlier, it could alter the slope and intercept of the plotted line, creating a poor observation. Removing such points creates a more generalized observation to all the other fitted points.
- Primarily that the assumption of linear functions relies on the data points, any extreme cases outside the normal will skew the output since these outliers will have a higher weight.

Q7. What car has the 4th highest LOF value? What attributes of this car showed up as univariate outliers according to our boxplots (IQR * 1.5 method)?

- Before removing LOF Values above 1.3, the honda civic had the 4th highest LOF out of all the cars. After removing, the 4th highest LOF score is the Fiat 128.
- Fiat: qsec
- Honda: wt

Q8. In LOF method, there is a hyperparameter, what is it? And what does it represent?

- The hyperparamater was the n_neighbors,and it defines how many points (neighbors) around each data point should be included in this comparison when calculating the LOF. Primarily controls the scope of the neighborhood that LOF uses to assess whether a point is an outlier by comparing its density with the densities of its neighbors.

Q9. Which outlier treatment worked best on this data set? [given the random_state=42 when splitting the data]

- Removing the outliers with LOF with Gradient Boosting gave an output with the best results in terms of R^2 with MSE. The R^2 was 0.916821 for linear, 0.873599 for SVM, and 0.976491 for Gradient Boosting. All of which were at least .10 higher than the other methods. The MSE for LOF was 5.113976 for linear, 7.771392 for SVM and 1.445399 for Gradient Boosting. All of which is the lowest value comparatively in all categories.

Q10. Which model is most likely to exhibit benign over-fitting? If we theoretically put these models into production and tested them on new (previously unseen) cars, how might we detect that over-fitting? Is it possible that what we previously treated as "outliers" would appear more "normal" in our data over time?

- The one model that we have that is likely to benign overfit would be the gradient boosting regressor. Having many splits could create an overfit scenario while having good performance.
- If we were detect overfitting, looking at the R^2 and MSE would be a good indication. Seeing a decrease in R^2 accuracy and a increase in MSE after introducing the new data could be an indication of overfitting.
- It depends on the data introduced, but yes it is possible for the oultiers to be considered normals, and even the some of the current data to be outliers. A concept drift would be the approach for this and occurs when the data distribution changes over time.